BDA Lab Questions

PART-I

1. Wordcount Map Reduce program using standalone Hadoop?

```
import java.io.IOException;

 import java.util.StringTokenizer;
import org.apache.hadoop.conf.Configuration;
 import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
 import org.apache.hadoop.mapreduce.Mapper;
 import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {

  public static class TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1); private
    Text word = new Text();

    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString()); while
      (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }


  public static class IntSumReducer
       extends Reducer<Text,IntWritable,Text,IntWritable> { private
    IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values, Context
                       context
                       ) throws IOException, InterruptedException { int
      sum = 0;
      for (IntWritable val : values) { sum
        += val.get();
      }
      result.set(sum);


      context.write(key, result);
    }
  }
  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

EXECUTION PROCEDURE:

1. Create one folder on Desktop-”WordCountTutorial”
    a. Paste WordCount.java file
    b. Create a folder named ”input_data”->Create input.txt file (enter some words)
    c. Create a folder name “tutorial_classes
2. Export HADOOP_CLASSPATH=$(hadoop classpath)
3. echo $HADOOP_CLASPATH
4. hadoop fs -mkdir /WordCountTutorial
5. hadoop fs -mkdir /WordCountTutorial/input
6. hadoop fs -put '/home/vignan/Desktop/WordCountTutorial/Input_Data/Input.txt' /WordCountTutorial/input

7. Change the current directory to the tutorial directory cd
'/home/vignan/Desktop/WordCountTutorial'
vignan@vignan-HP-Compaq-8200-Elite-SFF-PC:~/Desktop/WordCountTutorial$

8. Compile the java code
javac -classpath ${HADOOP_CLASSPATH} -d
'/home/vignan/Desktop/WordCountTutorial/tutorial_classes'
'/home/vignan/Desktop/WordCountTutorial/WordCount.java'

9. Put the output files in one JAR file
jar -cvf firstTutorial.jar -C tutorial_classes/ .

10. Run JAR file
Hadoop jar ‘/home/vignan/Desktop/WordCountTutorial/ firstTutorial.jar’ WordCount

/WordCountTutorial/input /WordCountTutorial/output

11. See the output
Hadoop dfs -cat /WordCountTutorial/Output/*

2. Implementation of Sort operation using MapReduce?

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
public class Sort{

	public static class SortMapper extends Mapper<LongWritable,Text,Text,Text>{
	Text comp_key = new Text();

		protected void map(LongWritable key, Text value, Context context) throws
IOException,InterruptedException{


	String[] token = value.toString().split(",");
	comp_key.set(token[0]);
	comp_key.set(token[1]);

context.write(comp_key,new Text(token[0]+"-"+token[1]+"-"+token[2]));
		}
	}


	public static class SortReducer extends Reducer<Text,Text,NullWritable,Text>{
		public void reduce(Text key, Iterable<Text> values, Context context) throws

IOException,InterruptedException{


		for(Text details:values){
			context.write(NullWritable.get(),details);
		}
	   }
  }
public static void main(String args[]) throws
IOException,InterruptedException,ClassNotFoundException{
        Configuration conf = new Configuration();
        Job job = new Job(conf);
        job.setJarByClass(Sort.class);
        job.setMapperClass(SortMapper.class);
        job.setReducerClass(SortReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true)?0:1);

  }

 }
```

3. Write a user define partitioner class for WordCount problem?

```
import java.io.IOException;
import java.util.*;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.util.*;


public class WordCount {

public static class Map extends MapReduceBase implements Mapper<LongWritable,
                    Text, Text, IntWritable> {

private final static IntWritable one = new IntWritable(1);
private Text word = new Text();

public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable>
                output, Reporter reporter) throws IOException {

    String line = value.toString();

    StringTokenizer tokenizer = new StringTokenizer(line);
    while (tokenizer.hasMoreTokens()) {
    word.set(tokenizer.nextToken());

      output.collect(word, one);
```

```
                }
              }
            }

public static class Reduce extends MapReduceBase implements Reducer<Text, IntWritable,
Text, IntWritable> {
public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text,
IntWritable> output, Reporter reporter) throws IOException {
                int sum = 0;
                while (values.hasNext()) {
                  sum += values.next().get();
                }
                output.collect(key, new IntWritable(sum));
              }
            }

public static void main(String[] args) throws Exception {
              JobConf conf = new JobConf(WordCount.class);
              conf.setJobName("wordcount");

              conf.setOutputKeyClass(Text.class);
              conf.setOutputValueClass(IntWritable.class);

              conf.setMapperClass(Map.class);
              conf.setCombinerClass(Reduce.class);
              conf.setReducerClass(Reduce.class);

              conf.setInputFormat(TextInputFormat.class);
              conf.setOutputFormat(TextOutputFormat.class);

              FileInputFormat.setInputPaths(conf, new Path(args[0]));
              FileOutputFormat.setOutputPath(conf, new Path(args[1]));

              JobClient.runJob(conf);
            }
          }
```

4. Write a MapReduce program to calculate employee salary of each department in the university?

```
import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;

import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
public class Salary

{

public static class SalaryMapper extends Mapper <LongWritable, Text, Text, IntWritable>

{

public void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException

{

String[] token = value.toString().split(",");
int s = Integer.parseInt(token[2]);
IntWritable sal = new IntWritable();
sal.set(s);

context.write(new Text(token[1]),sal);

}

}

public static class SalaryReducer extends Reducer<Text, IntWritable, Text, IntWritable>

{

private IntWritable result = new IntWritable();

public void reduce(Text key, Iterable<IntWritable> values, Context context ) throws
IOException, InterruptedException

{

int sum = 0;

for (IntWritable val : values) {
sum += val.get();

}

result.set(sum);
context.write(key,result);

}

}
public static void main(String[] args) throws Exception {
Configuration conf = new Configuration();

Job job = Job.getInstance(conf, "Salary");
job.setJarByClass(Salary.class);
job.setMapperClass(SalaryMapper.class);

job.setReducerClass(SalaryReducer.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
FileInputFormat.addInputPath(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));
System.exit(job.waitForCompletion(true) ? 0 : 1);

}

}
```

5. Write a MapReduce program to search an employee name in the following data?

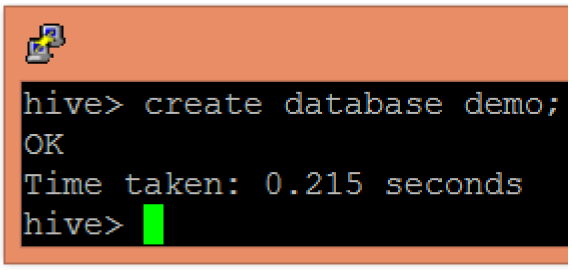
```
import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;

import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
public class Name

{

public static class NameMapper extends Mapper <LongWritable, Text, Text, IntWritable>

{

public void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException

{

String[] token = value.toString().split(",");
int s = Integer.parseInt(token[2]);

 String Writable name = new String
Writable(); name.set(s);

context.write(new Text(token[1]),name);

}
}

public static class NameReducer extends Reducer<Text, StringWritable, Text, StringWritable>

{

private IntWritable result = new IntWritable();

public void reduce(Text key, Iterable<IntWritable> values, Context context ) throws
IOException, InterruptedException

{

int sum = 0;

for (IntWritable val : values) {
sum += val.get();

}

result.set(sum);
context.write(key,result);

}

}

public static void main(String[] args) throws Exception {
Configuration conf = new Configuration();

Job job = Job.getInstance(conf, "Salary");
job.setJarByClass(Salary.class);
job.setMapperClass(SalaryMapper.class);

job.setReducerClass(SalaryReducer.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
FileInputFormat.addInputPath(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));
System.exit(job.waitForCompletion(true) ? 0 : 1);

}

}
```

PART-II

6. Creation of Database, External and Internal tables using Hive?

Creation of Database:

```
hive> create database demo;
```

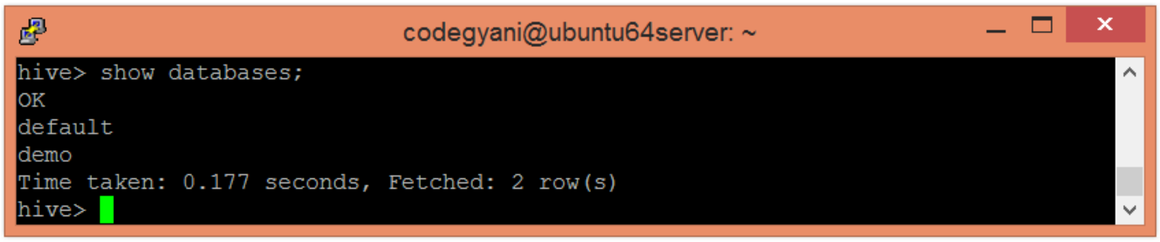


```
hive> create database demo;
OK
Time taken: 0.215 seconds
hive>
```

So, a new database is created.

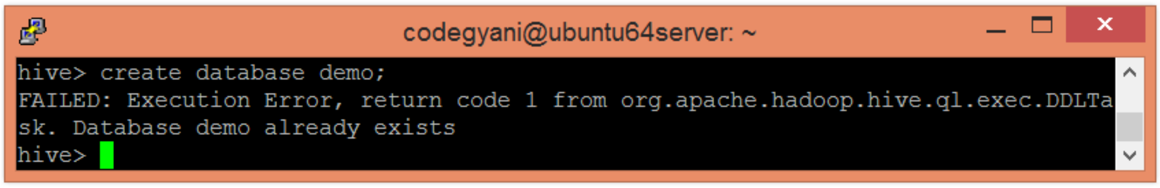
- Let's check the existence of a newly created database.

```
hive> show databases;
```



```
hive> show databases;
OK
default
demo
Time taken: 0.177 seconds, Fetched: 2 row(s)
hive>
```

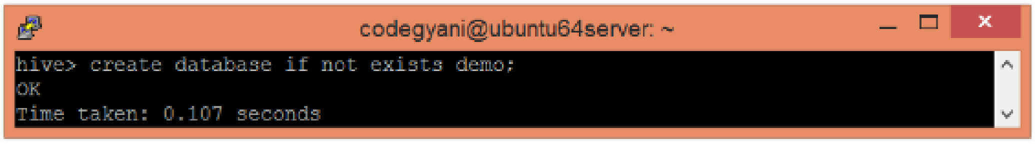
- Each database must contain a unique name. If we create two databases with the same name, the following error generates: -



```
hive> create database demo;
FAILED: Execution Error, return code 1 from org.apache.hadoop.hive.ql.exec.DDLTa
sk. Database demo already exists
hive>
```

- If we want to suppress the warning generated by Hive on creating the database with the same name, follow the below command: -

```
hive> create a database if not exists demo;
```

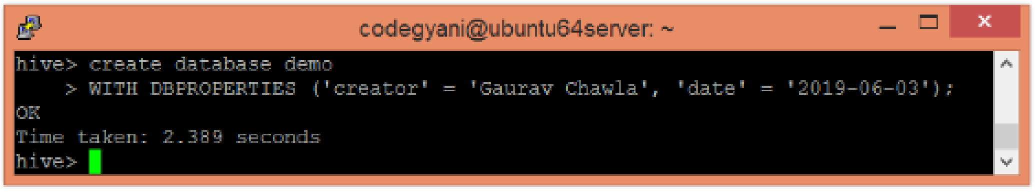


codegyani@ubuntu64server: ~

```
hive> create database if not exists demo;
OK
Time taken: 0.107 seconds
```

- Hive also allows assigning properties with the database in the form of key-value pair.

```
hive>create the database demo
    >WITH DBPROPERTIES ('creator' = 'Gaurav Chawla', 'date' = '2019-06-03');
```

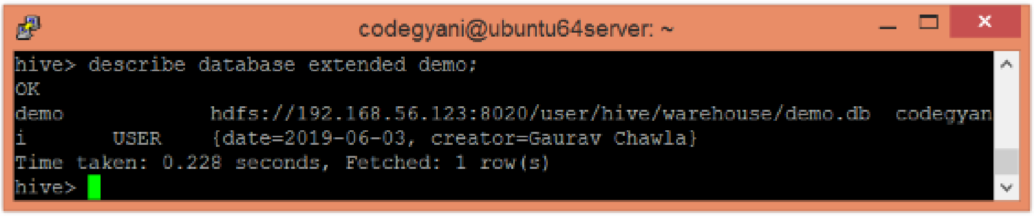


codegyani@ubuntu64server: ~

```
hive> create database demo
    > WITH DBPROPERTIES ('creator' = 'Gaurav Chawla', 'date' = '2019-06-03');
OK
Time taken: 2.389 seconds
hive>
```

- Let's retrieve the information associated with the database.

```
hive> describe database extended demo;
```



codegyani@ubuntu64server: ~

```
hive> describe database extended demo;
OK
demo            hdfs://192.168.56.123:8020/user/hive/warehouse/demo.db  codegyan
i       USER    {date=2019-06-03, creator=Gaurav Chawla}
Time taken: 0.228 seconds, Fetched: 1 row(s)
hive>
```

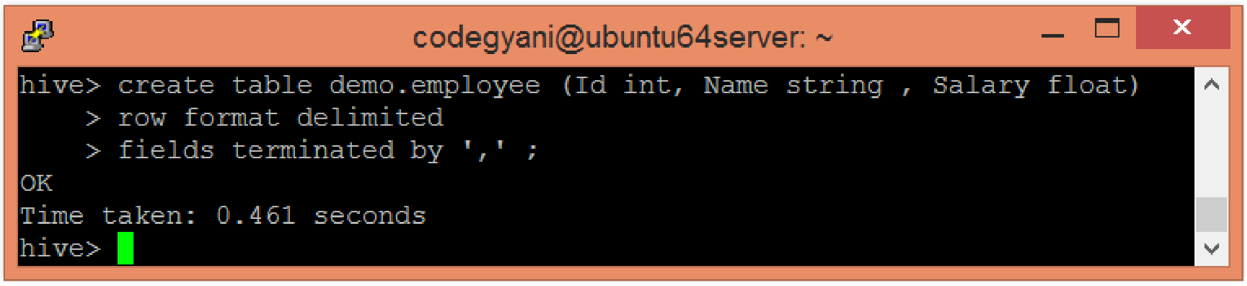
External tables:

Internal Table

The internal tables are also called managed tables as the lifecycle of their data is controlled by the Hive. By default, these tables are stored in a subdirectory under the directory defined by hive.metastore.warehouse.dir (i.e. /user/hive/warehouse). The internal tables are not flexible enough to share with other tools like Pig. If we try to drop the internal table, Hive deletes both table schema and data.

- Let's create an internal table by using the following command:-

1. hive> create table demo.employee (Id int, Name string , Salary float)
2. row format delimited
3. fields terminated by ',' ;

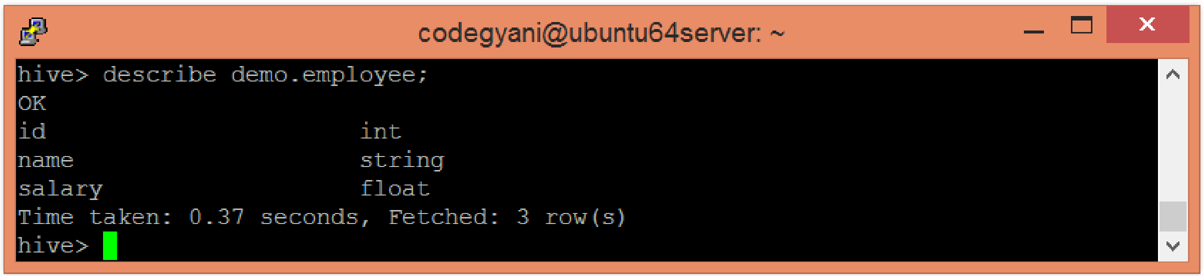


```
hive> create table demo.employee (Id int, Name string , Salary float)
    > row format delimited
    > fields terminated by ',' ;
OK
Time taken: 0.461 seconds
hive>
```

Here, the command also includes the information that the data is separated by ','.

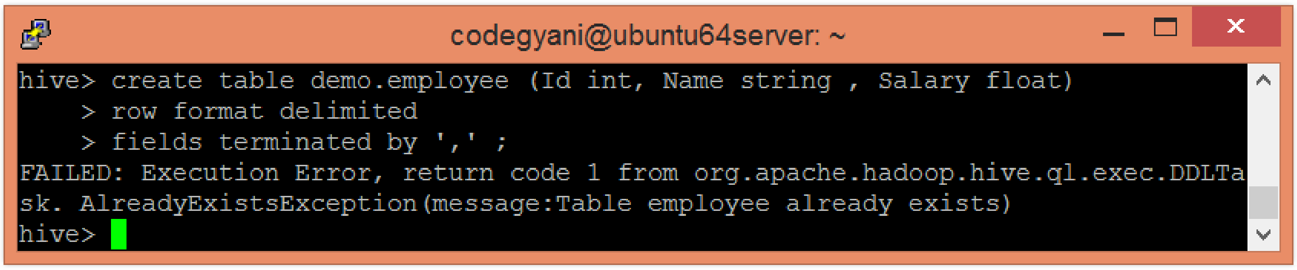
- Let's see the metadata of the created table by using the following command:-

1. hive> describe demo.employee



```
hive> describe demo.employee;
OK
id                      int
name                    string
salary                  float
Time taken: 0.37 seconds, Fetched: 3 row(s)
hive>
```

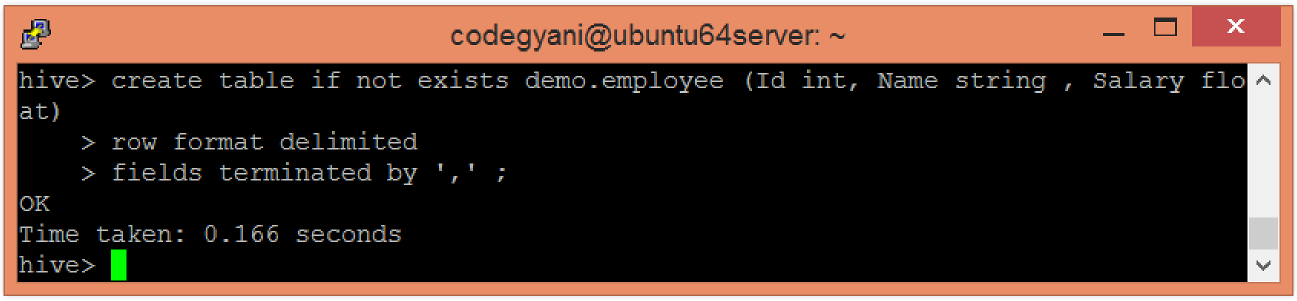
- Let's see the result when we try to create the existing table again.



```
hive> create table demo.employee (Id int, Name string , Salary float)
    > row format delimited
    > fields terminated by ',' ;
FAILED: Execution Error, return code 1 from org.apache.hadoop.hive.ql.exec.DDLTa
sk. AlreadyExistsException(message:Table employee already exists)
hive>
```

In such a case, the exception occurs. If we want to ignore this type of exception, we can use **if not exists** command while creating the table.

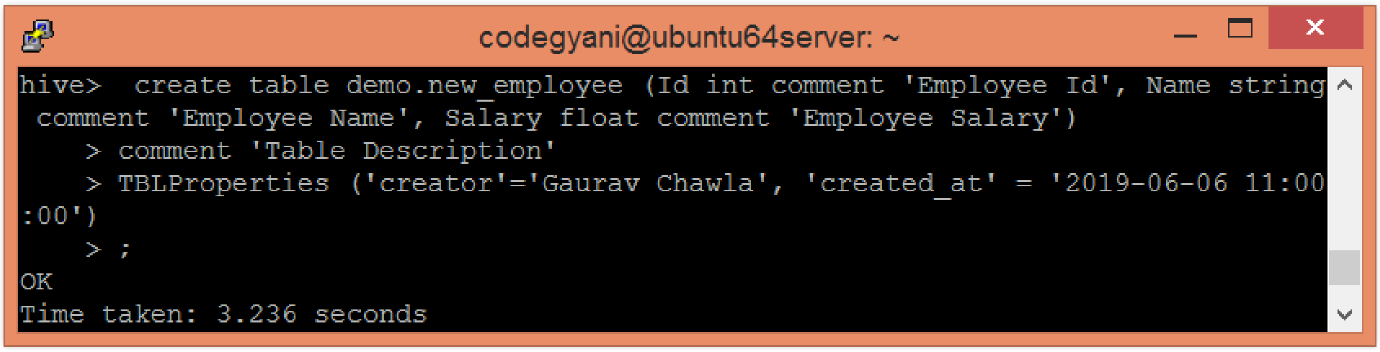
1. hive> create table if not exists demo.employee (Id int, Name string , Salary float)
2. row format delimited
3. fields terminated by ',' ;



```
hive> create table if not exists demo.employee (Id int, Name string , Salary flo
at)
    > row format delimited
    > fields terminated by ',' ;
OK
Time taken: 0.166 seconds
hive>
```

- While creating a table, we can add the comments to the columns and can also define the table properties.

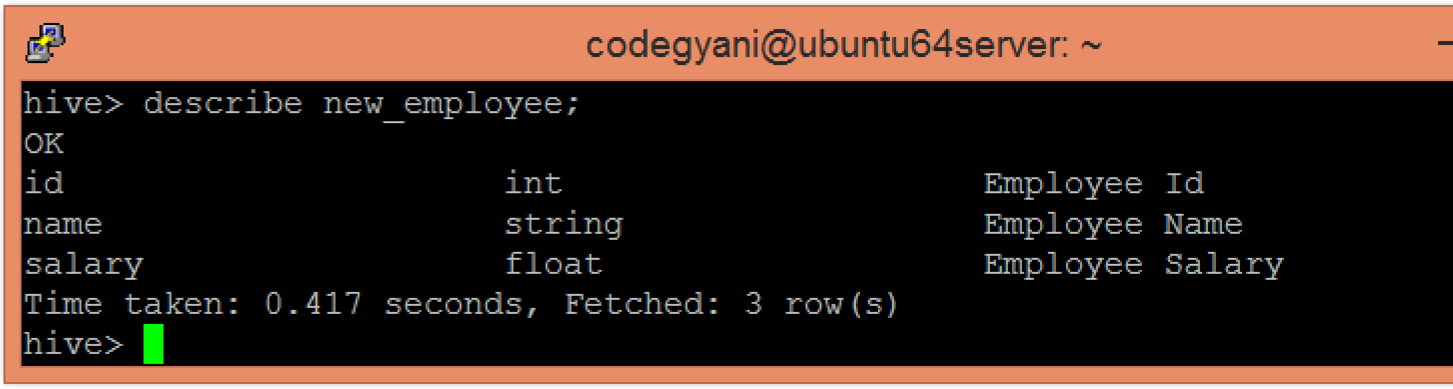
1. hive> create table demo.new_employee (Id int comment 'Employee Id', Name string comment 'Employee Name', Salary float comment 'Employee Salary')
2. comment 'Table Description'
3. TBLProperties ('creator'='Gaurav Chawla', 'created_at' = '2019-06-06 11:00:00');



```
hive>  create table demo.new_employee (Id int comment 'Employee Id', Name string
 comment 'Employee Name', Salary float comment 'Employee Salary')
    > comment 'Table Description'
    > TBLProperties ('creator'='Gaurav Chawla', 'created_at' = '2019-06-06 11:00
:00')
    > ;
OK
Time taken: 3.236 seconds
```

- Let's see the metadata of the created table by using the following command: -

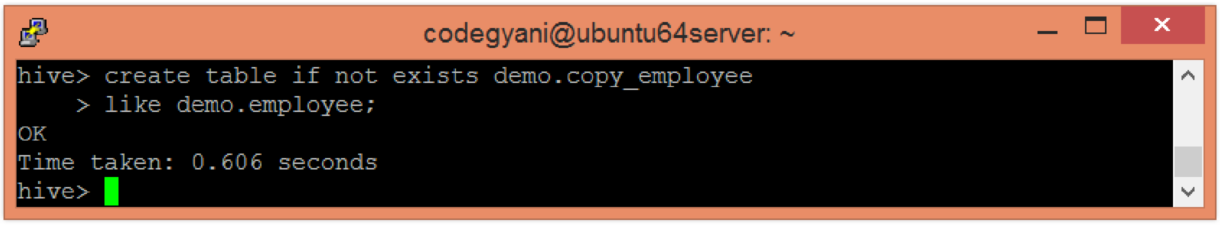
1. hive> describe new_employee;



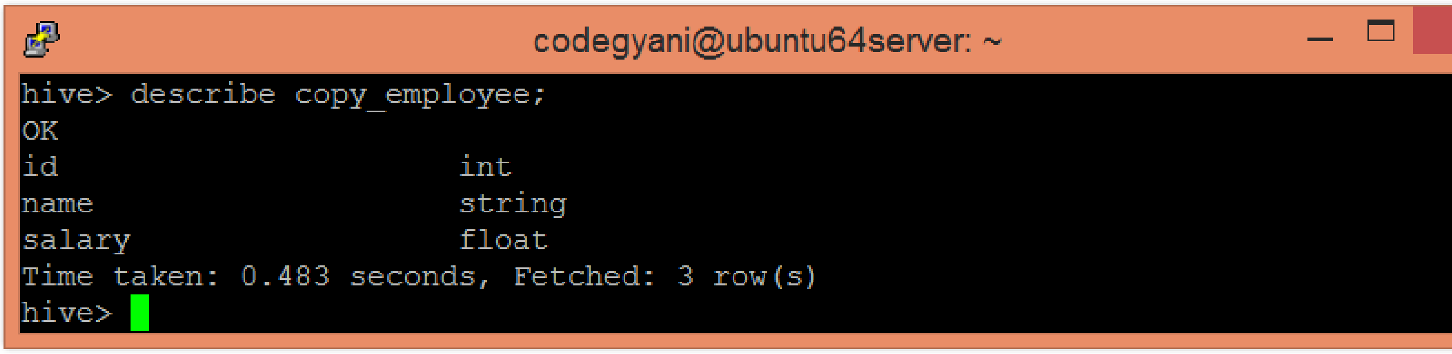
```
hive> describe new_employee;
OK
id                      int                     Employee Id
name                    string                  Employee Name
salary                  float                   Employee Salary
Time taken: 0.417 seconds, Fetched: 3 row(s)
hive>
```

- Hive allows creating a new table by using the schema of an existing table.

1. hive> create table if not exists demo.copy_employee
2. like demo.employee;



```
hive> create table if not exists demo.copy_employee
    > like demo.employee;
OK
Time taken: 0.606 seconds
hive>
```



```
hive> describe copy_employee;
OK
id                      int
name                    string
salary                  float
Time taken: 0.483 seconds, Fetched: 3 row(s)
hive>
```

Here, we can say that the new table is a copy of an existing table.

External Table

The external table allows us to create and access a table and a data externally. The **external** keyword is used to specify the external table, whereas the **location** keyword is used to determine the location of loaded data.

As the table is external, the data is not present in the Hive directory. Therefore, if we try to drop the table, the metadata of the table will be deleted, but the data still exists.

To create an external table, follow the below steps: -

- Let's create a directory on HDFS by using the following command: -

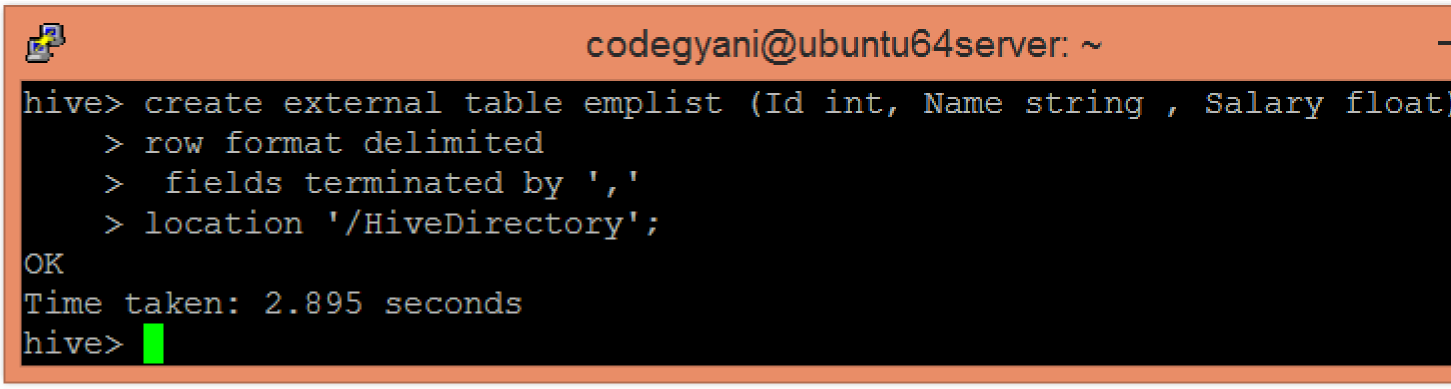
1. hdfs dfs -mkdir /HiveDirectory

- Now, store the file on the created directory.

1. hdfs dfs -put hive/emp_details /HiveDirectory

- Let's create an external table using the following command: -

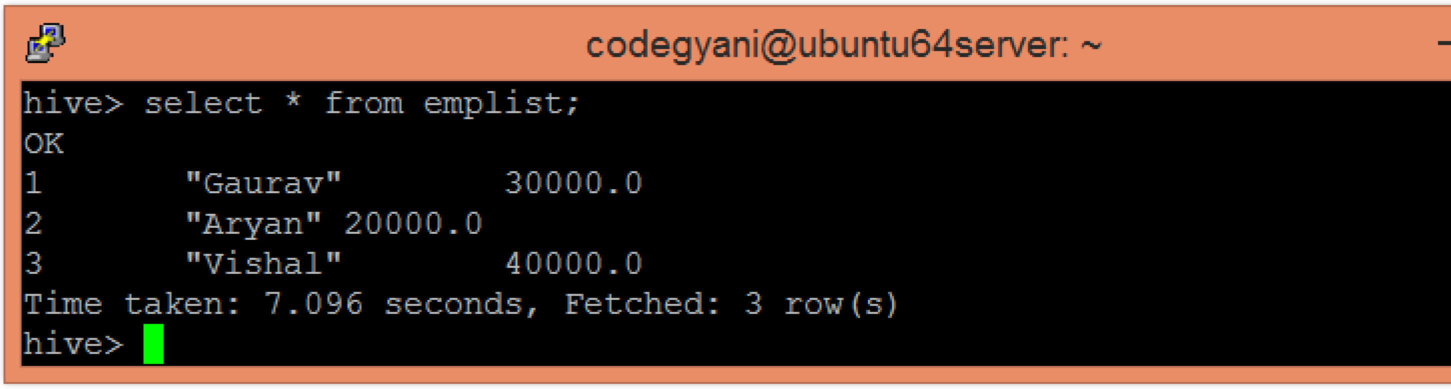
1. hive> create external table emplist (Id int, Name string , Salary float)
2. row format delimited
3. fields terminated by ','
4. location '/HiveDirectory';



```
hive> create external table emplist (Id int, Name string , Salary float)
    > row format delimited
    >  fields terminated by ','
    > location '/HiveDirectory';
OK
Time taken: 2.895 seconds
hive>
```

- Now, we can use the following command to retrieve the data: -

1. select * from emplist;



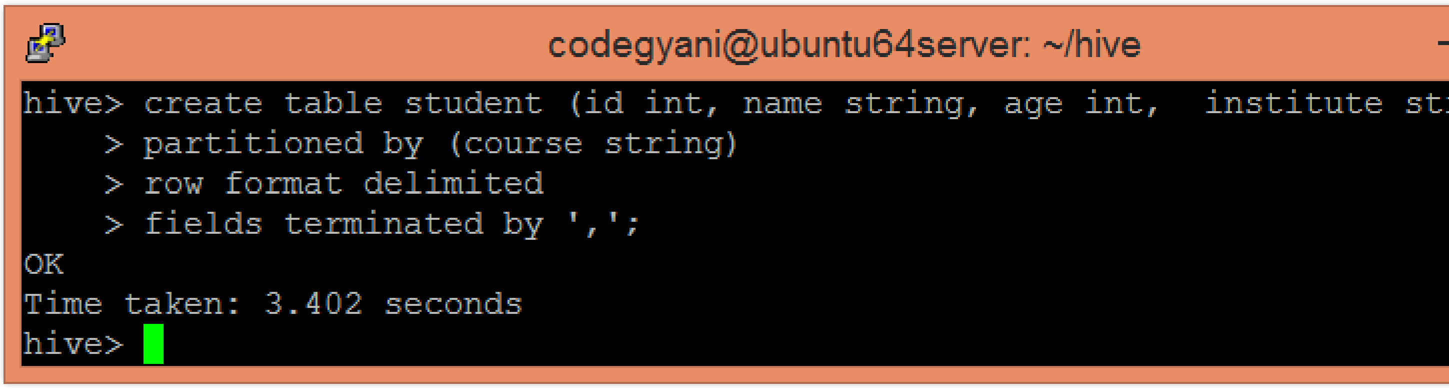
```
hive> select * from emplist;
OK
1       "Gaurav"        30000.0
2       "Aryan" 20000.0
3       "Vishal"        40000.0
Time taken: 7.096 seconds, Fetched: 3 row(s)
hive>
```

7. **Creation of Static partition, Dynamic partition, and Buckets using Hive?**

Static Partitioning

- Create the table and provide the partitioned columns by using the following command: -

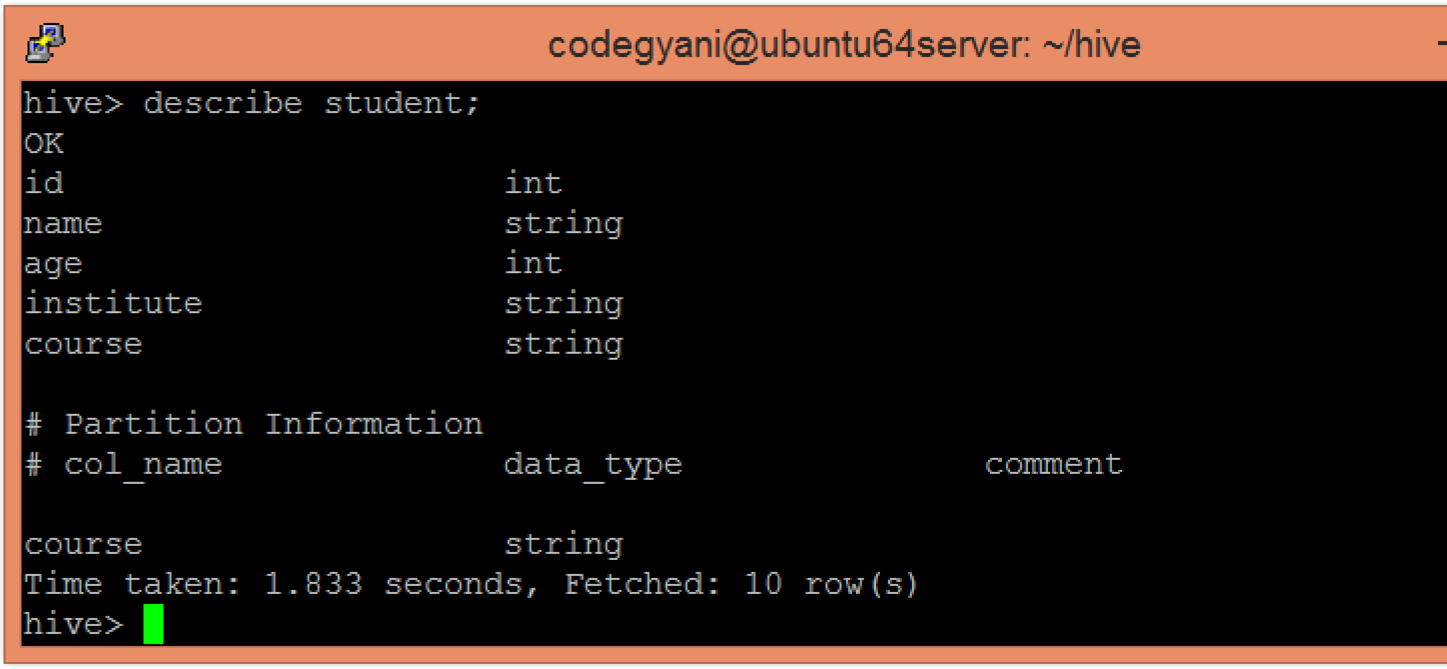
1. hive> create table student (id int, name string, age int, institute string)
2. partitioned by (course string)
3. row format delimited
4. fields terminated by ',';



```
hive> create table student (id int, name string, age int,  institute st
    > partitioned by (course string)
    > row format delimited
    > fields terminated by ',';
OK
Time taken: 3.402 seconds
hive>
```

- Let's retrieve the information associated with the table.

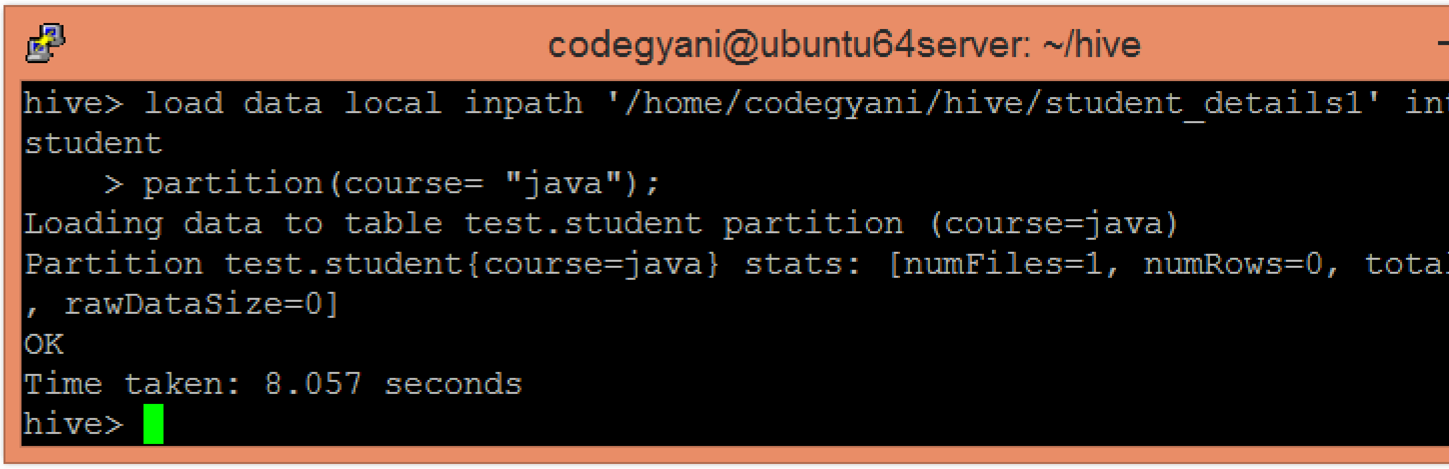
1. hive> describe student;



```
codegyani@ubuntu64server: ~/hive
hive> describe student;
OK
id                      int
name                    string
age                     int
institute               string
course                  string

# Partition Information
# col_name              data_type               comment

course                  string
Time taken: 1.833 seconds, Fetched: 10 row(s)
hive>
```

- Load the data into the table and pass the values of partition columns with it by using the following command: -

1. hive> load data local inpath '/home/codegyani/hive/student_details1' into table student
2. partition(course= "java");

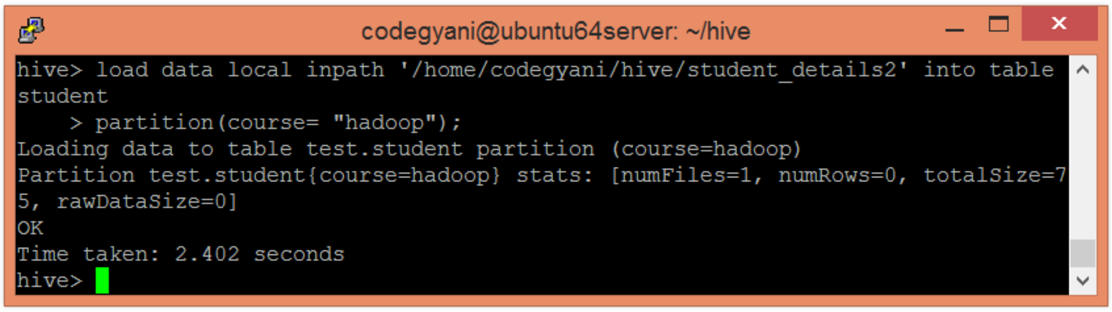


```
codegyani@ubuntu64server: ~/hive
hive> load data local inpath '/home/codegyani/hive/student_details1' in
student
    > partition(course= "java");
Loading data to table test.student partition (course=java)
Partition test.student{course=java} stats: [numFiles=1, numRows=0, tota
, rawDataSize=0]
OK
Time taken: 8.057 seconds
hive>
```

Here, we are partitioning the students of an institute based on courses.

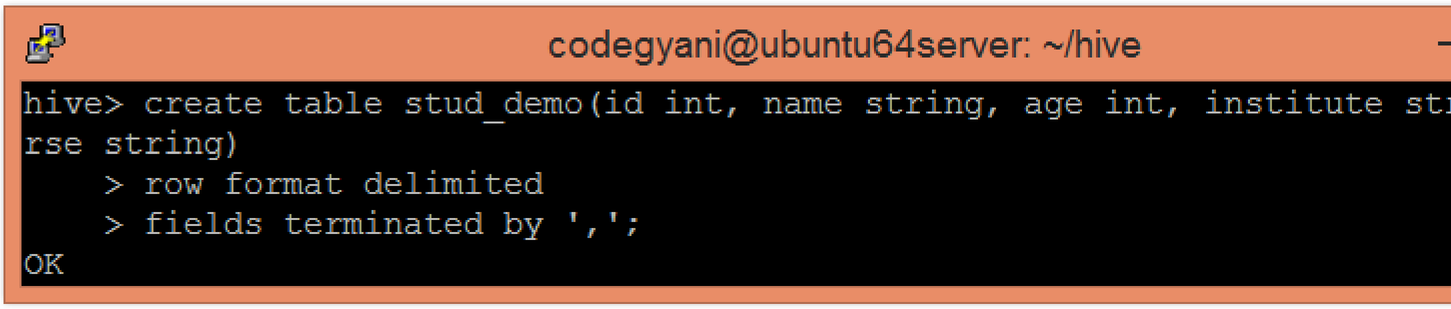
- Load the data of another file into the same table and pass the values of partition columns with it by using the following command: -

1. hive> load data local inpath '/home/codegyani/hive/student_details2' into table student
2. partition(course= "hadoop");


```
hive> load data local inpath '/home/codegyani/hive/student_details2' into table
student
    > partition(course= "hadoop");
Loading data to table test.student partition (course=hadoop)
Partition test.student{course=hadoop} stats: [numFiles=1, numRows=0, totalSize=7
5, rawDataSize=0]
OK
Time taken: 2.402 seconds
hive>
```

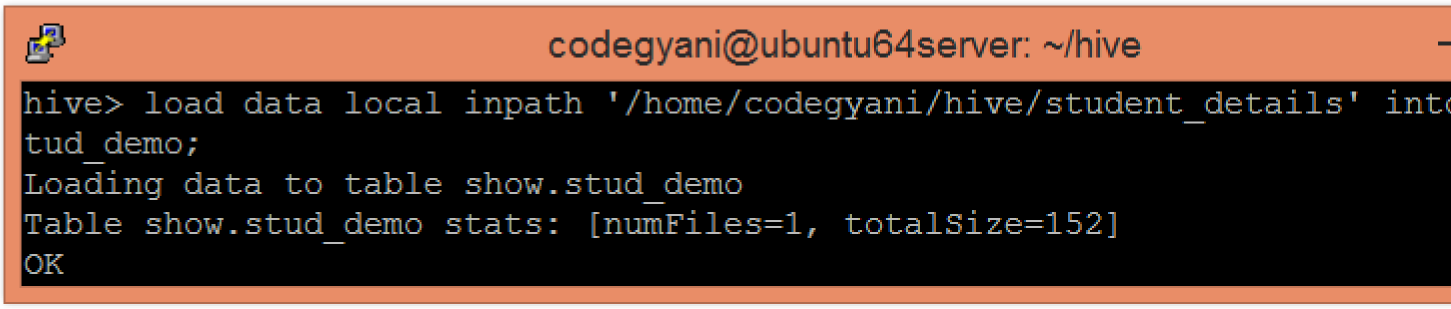
Dynamic Partitioning in hive

1. hive> create table stud_demo(id int, name string, age int, institute string, course string)
2. row format delimited
3. fields terminated by ',';


```
hive> create table stud_demo(id int, name string, age int, institute str
rse string)
    > row format delimited
    > fields terminated by ',';
OK
```

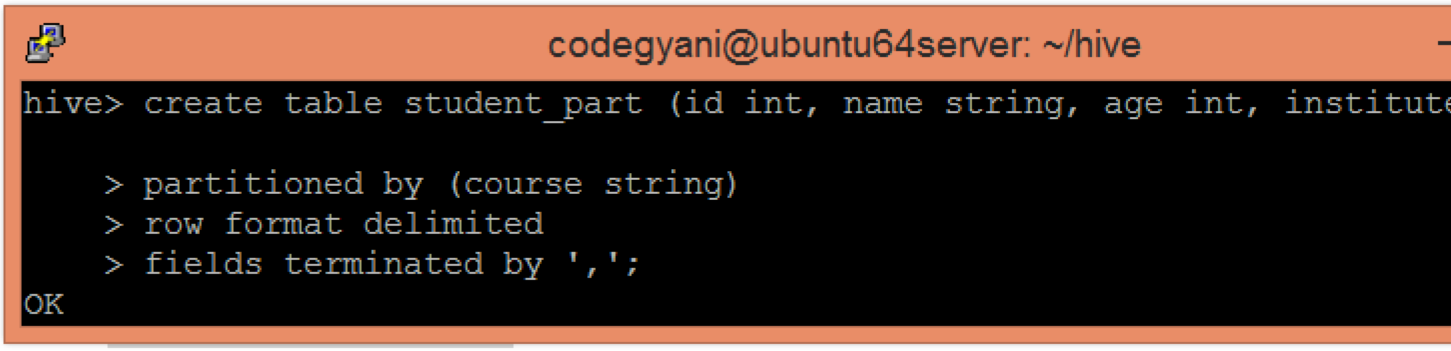
- Now, load the data into the table.

1. hive> load data local inpath '/home/codegyani/hive/student_details' into table stud_demo;


```
hive> load data local inpath '/home/codegyani/hive/student_details' into
tud_demo;
Loading data to table show.stud_demo
Table show.stud_demo stats: [numFiles=1, totalSize=152]
OK
```

- Create a partition table by using the following command: -

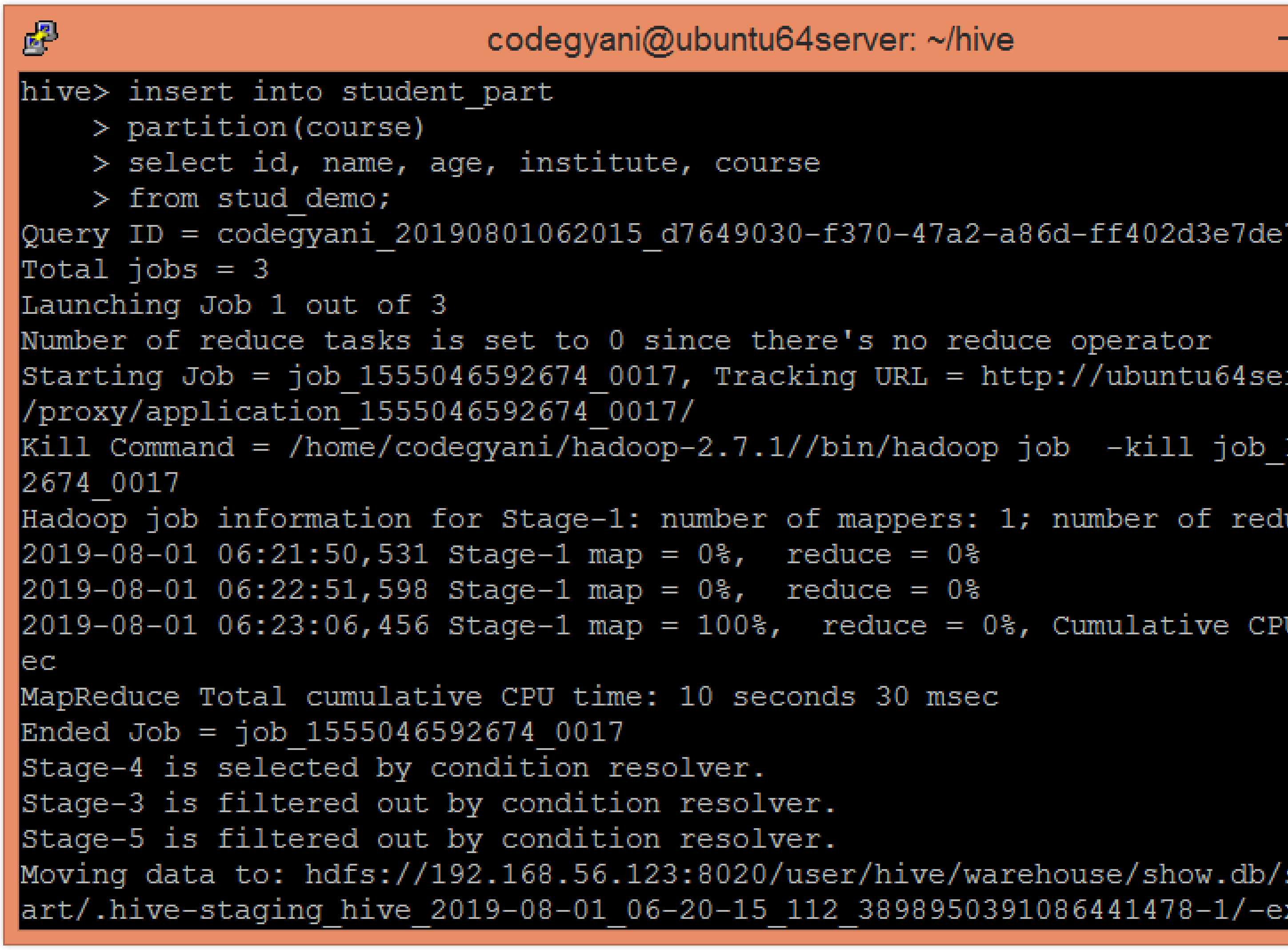
1. hive> create table student_part (id int, name string, age int, institute string)
2. partitioned by (course string)
3. row format delimited
4. fields terminated by ',';



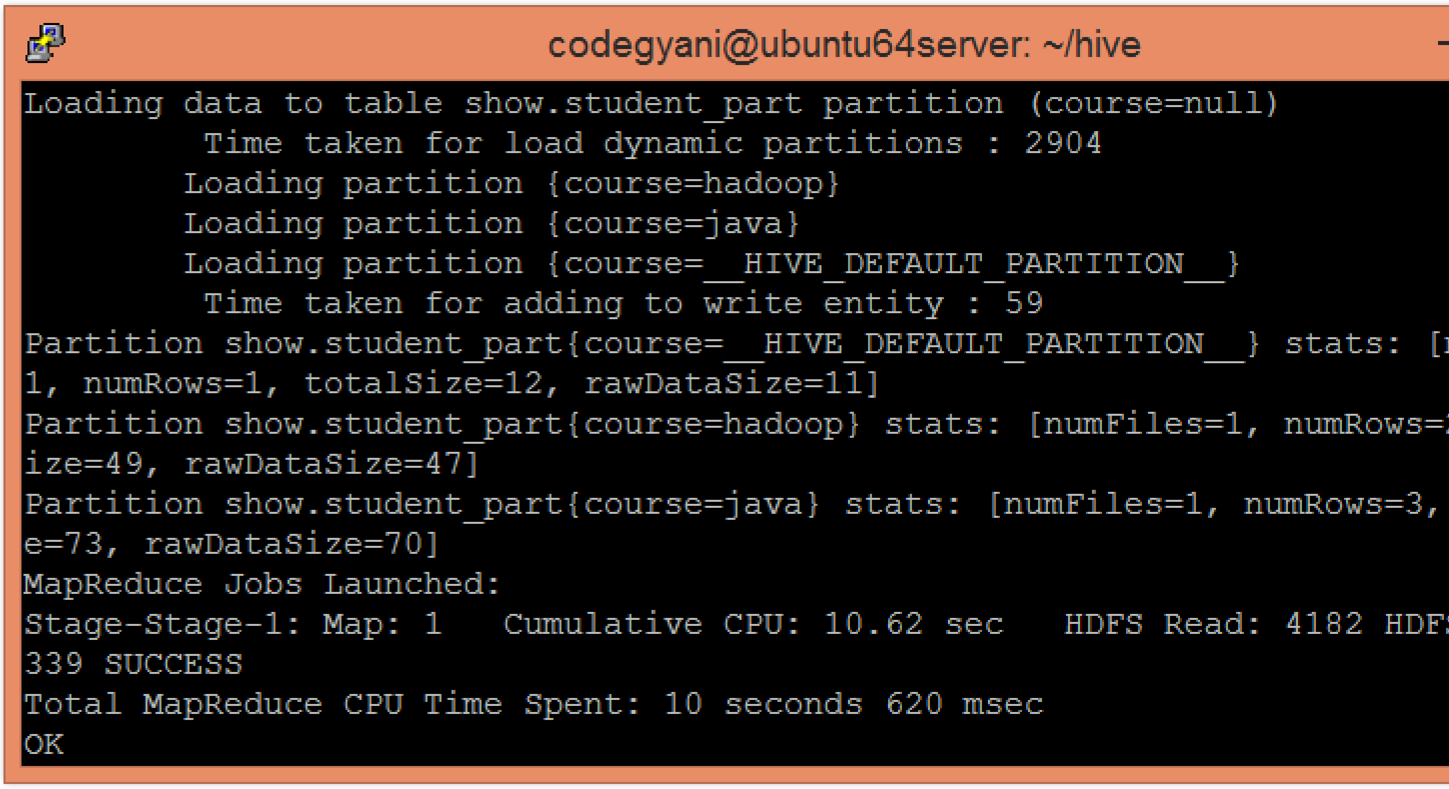
```
codegyani@ubuntu64server: ~/hive
hive> create table student_part (id int, name string, age int, institute

    > partitioned by (course string)
    > row format delimited
    > fields terminated by ',';
OK
```

- Now, insert the data of dummy table into the partition table.

1. hive> insert into student_part
2. partition(course)
3. select id, name, age, institute, course
4. from stud_demo;



```
codegyani@ubuntu64server: ~/hive
hive> insert into student_part
    > partition(course)
    > select id, name, age, institute, course
    > from stud_demo;
Query ID = codegyani_20190801062015_d7649030-f370-47a2-a86d-ff402d3e7de
Total jobs = 3
Launching Job 1 out of 3
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1555046592674_0017, Tracking URL = http://ubuntu64se
/proxy/application_1555046592674_0017/
Kill Command = /home/codegyani/hadoop-2.7.1//bin/hadoop job  -kill job_
2674_0017
Hadoop job information for Stage-1: number of mappers: 1; number of red
2019-08-01 06:21:50,531 Stage-1 map = 0%,  reduce = 0%
2019-08-01 06:22:51,598 Stage-1 map = 0%,  reduce = 0%
2019-08-01 06:23:06,456 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU
ec
MapReduce Total cumulative CPU time: 10 seconds 30 msec
Ended Job = job_1555046592674_0017
Stage-4 is selected by condition resolver.
Stage-3 is filtered out by condition resolver.
Stage-5 is filtered out by condition resolver.
Moving data to: hdfs://192.168.56.123:8020/user/hive/warehouse/show.db/
art/.hive-staging_hive_2019-08-01_06-20-15_112_3898950391086441478-1/-e
```



```
Loading data to table show.student_part partition (course=null)
         Time taken for load dynamic partitions : 2904
        Loading partition {course=hadoop}
        Loading partition {course=java}
        Loading partition {course=__HIVE_DEFAULT_PARTITION__}
         Time taken for adding to write entity : 59
Partition show.student_part{course=__HIVE_DEFAULT_PARTITION__} stats: [n
1, numRows=1, totalSize=12, rawDataSize=11]
Partition show.student_part{course=hadoop} stats: [numFiles=1, numRows=
ize=49, rawDataSize=47]
Partition show.student_part{course=java} stats: [numFiles=1, numRows=3,
e=73, rawDataSize=70]
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1   Cumulative CPU: 10.62 sec   HDFS Read: 4182 HDF
339 SUCCESS
Total MapReduce CPU Time Spent: 10 seconds 620 msec
OK
```

8. **Practice of Hive Query Language operations? (DDL, DML, Group by, Having, Joins, Arithmetic operations, Relational operations)**

Introduction to Hive DDL commands

Hive DDL commands are the statements used for defining and changing the structure of a table or database in Hive. It is used to build or modify the tables and other objects in the database.

The several types of Hive DDL commands are:

1. CREATE
2. SHOW
3. DESCRIBE
4. USE
5. DROP
6. ALTER
7. TRUNCATE

Table-1 Hive DDL commands

DDL Command	Use With
CREATE	Database, Table
SHOW	Databases, Tables, Table Properties, Partitions, Functions, Index
DESCRIBE	Database, Table, view
USE	Database
DROP	Database, Table
ALTER	Database, Table
TRUNCATE	Table

Remaining answers:

Java t point:

https://www.javatpoint.com/hive

9. **Write Pig Latin scripts to perform Load, Join, Foreach, Sort, Group, Join, Project, Filter, Order by, Cross, Limitoperations on the data.**

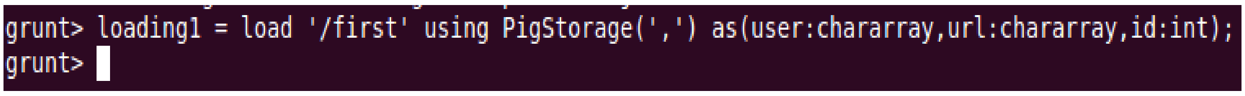
Relational Operators:

Relational operators are the main tools Pig Latin provides to operate on the data. It allows you to transform the data by sorting, grouping, joining, projecting and filtering. This section covers the basic relational operators.

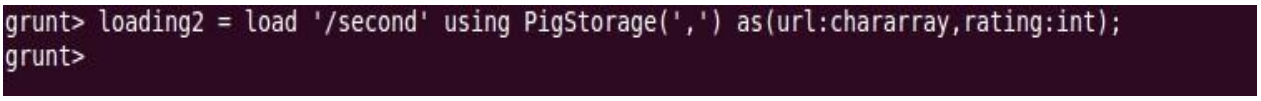
LOAD:

LOAD operator is used to load data from the file system or HDFS storage into a Pig relation.

In this example, the Load operator loads data from file 'first' to form relation 'loading1'. The field names are user, url, id.



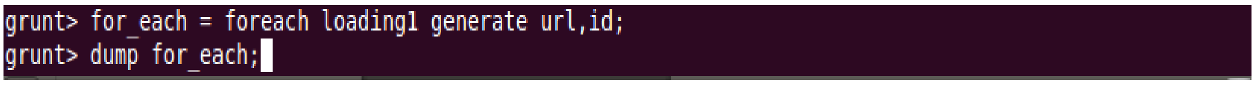
```
grunt> loading1 = load '/first' using PigStorage(',') as(user:chararray,url:chararray,id:int);
grunt>
```



```
grunt> loading2 = load '/second' using PigStorage(',') as(url:chararray,rating:int);
grunt>
```

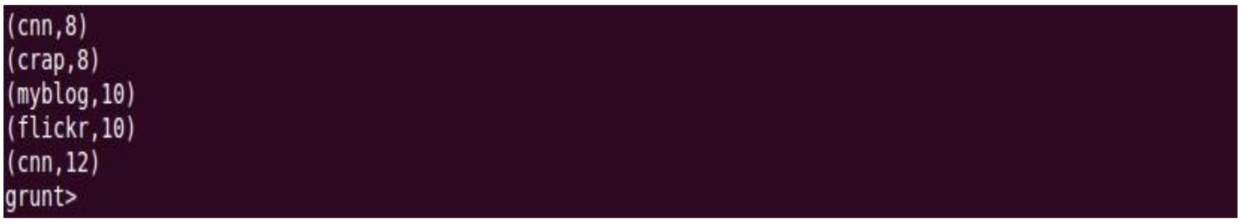
FOREACH:

This operator generates data transformations based on columns of data. It is used to add or remove fields from a relation. Use FOREACH-GENERATE operation to work with columns of data.



```
grunt> for_each = foreach loading1 generate url,id;
grunt> dump for_each;
```

FOREACH Result:

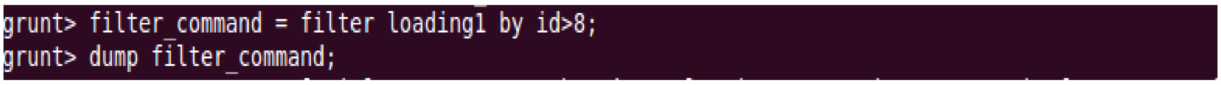


```
(cnn,8)
(crap,8)
(myblog,10)
(flickr,10)
(cnn,12)
grunt>
```

FILTER:

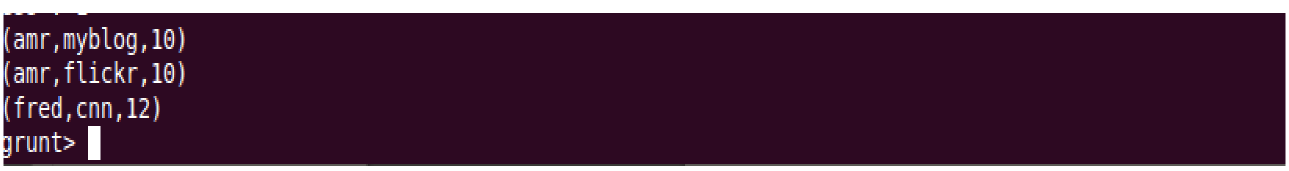
This operator selects tuples from a relation based on a condition.

In this example, we are filtering the record from ‘loading1’ when the condition ‘id’ is greater than 8.



```
grunt> filter_command = filter loading1 by id>8;
grunt> dump filter_command;
```

FILTER Result:

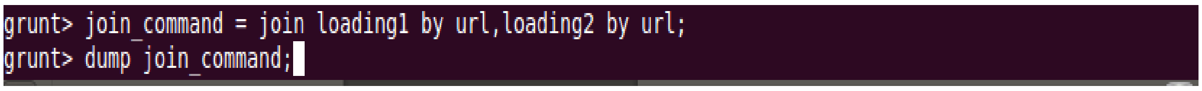


```
(amr,myblog,10)
(amr,flickr,10)
(fred,cnn,12)
grunt>
```

JOIN:

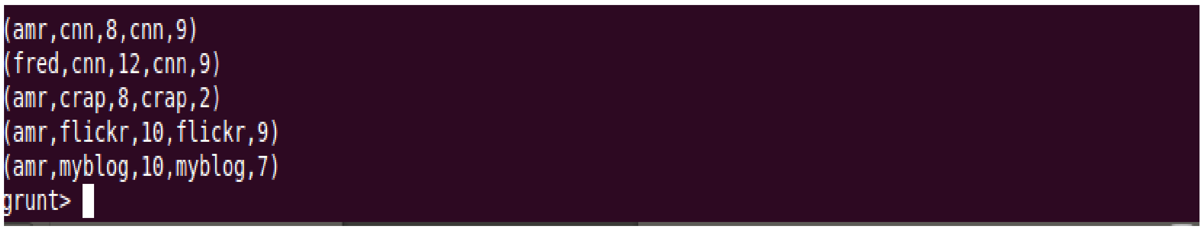
JOIN operator is used to perform an inner, equijoin join of two or more relations based on common field values. The JOIN operator always performs an inner join. Inner joins ignore null keys, so it makes sense to filter them out before the join.

In this example, join the two relations based on the column ‘url’ from ‘loading1’ and ‘loading2’.



```
grunt> join_command = join loading1 by url,loading2 by url;
grunt> dump join_command;
```

JOIN Result:

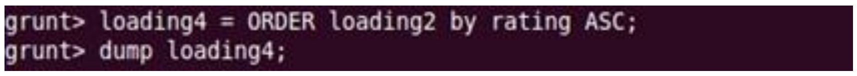


```
(amr,cnn,8,cnn,9)
(fred,cnn,12,cnn,9)
(amr,crap,8,crap,2)
(amr,flickr,10,flickr,9)
(amr,myblog,10,myblog,7)
grunt>
```

ORDER BY:

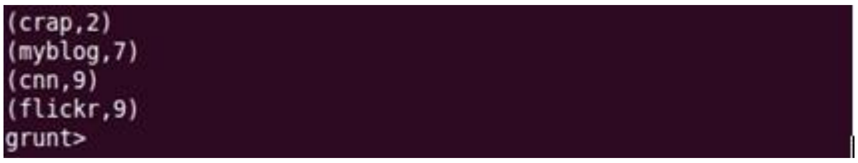
Order By is used to sort a relation based on one or more fields. You can do sorting in ascending or descending order using ASC and DESC keywords.

In below example, we are sorting data in loading2 in ascending order on ratings field.



```
grunt> loading4 = ORDER loading2 by rating ASC;
grunt> dump loading4;
```

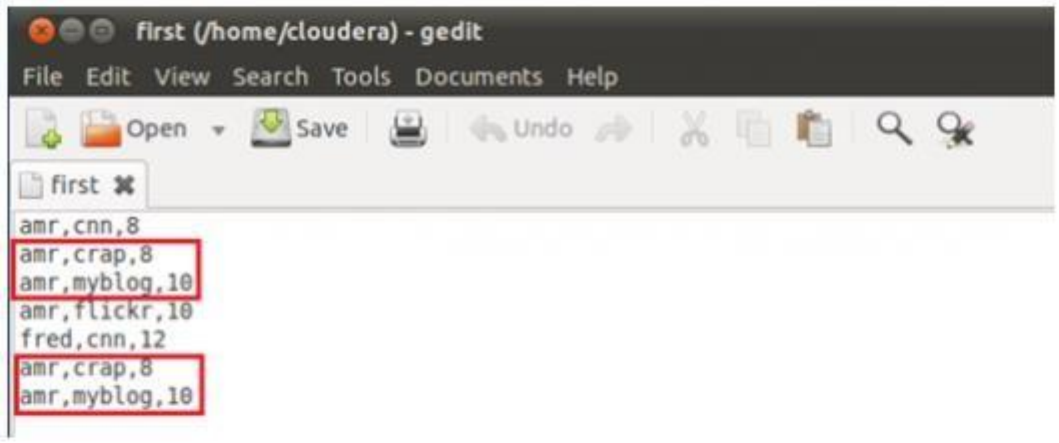
ORDER BY Result:



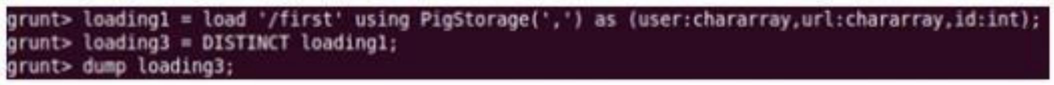
```
(crap,2)
(myblog,7)
(cnn,9)
(flickr,9)
grunt>
```

DISTINCT:

Distinct removes duplicate tuples in a relation.Lets take an input file as below, which has **amr,crap,8** and **amr,myblog,10**twice in the file. When we apply distinct on the data in this file, duplicate entries are removed.

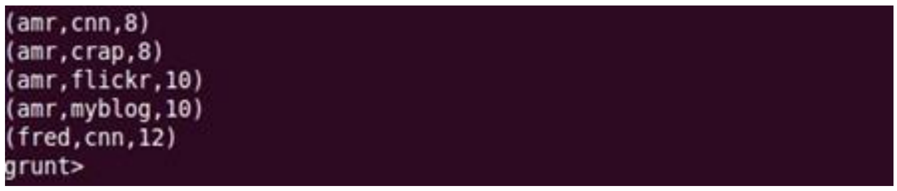

first (/home/cloudera) - gedit
File Edit View Search Tools Documents Help
Open
Save
Undo
first

```
amr,cnn,8
amr,crap,8
amr,myblog,10
amr,flickr,10
fred,cnn,12
amr,crap,8
amr,myblog,10
```



```
grunt> loading1 = load '/first' using PigStorage(',') as (user:chararray,url:chararray,id:int);
grunt> loading3 = DISTINCT loading1;
grunt> dump loading3;
```

DISTINCT Result:

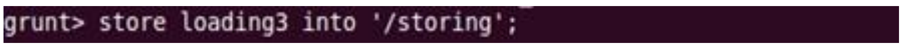


```
(amr,cnn,8)
(amr,crap,8)
(amr,flickr,10)
(amr,myblog,10)
(fred,cnn,12)
grunt>
```

STORE:

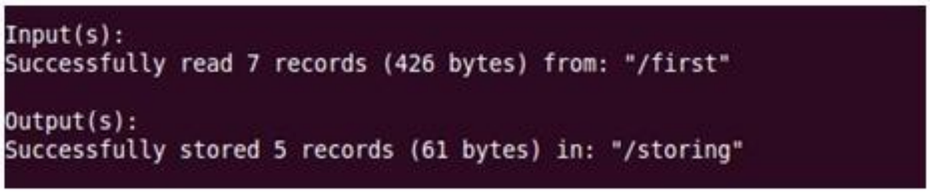
Store is used to save results to the file system.

Here we are saving **loading3** data into a file named **storing** on HDFS.

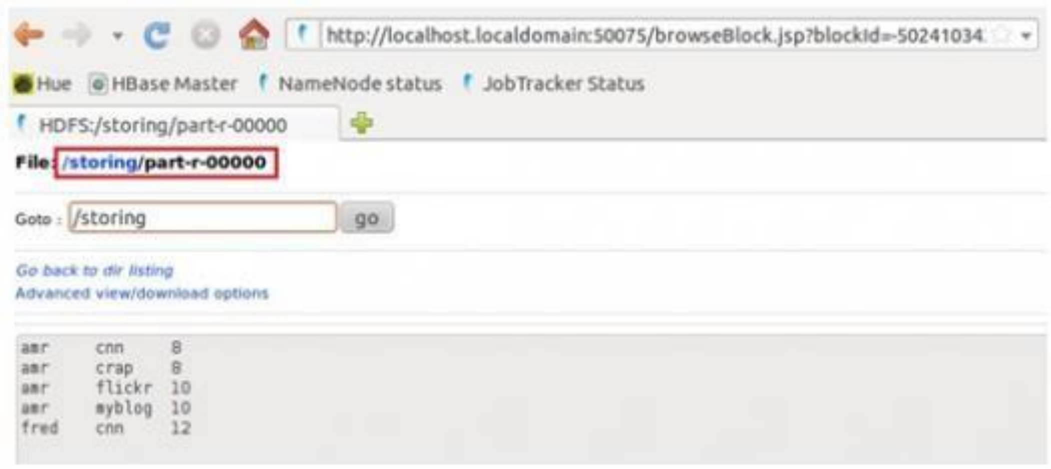


```
grunt> store loading3 into '/storing';
```

STORE Result:



```
Input(s):
Successfully read 7 records (426 bytes) from: "/first"

Output(s):
Successfully stored 5 records (61 bytes) in: "/storing"
```



http://localhost.localdomain:50075/browseBlock.jsp?blockId=-50241034
Hue
HBase Master
NameNode status
JobTracker Status
HDFS:/storing/part-r-00000
go
Go back to dir listing
Advanced view/download options

File: /storing/part-r-00000

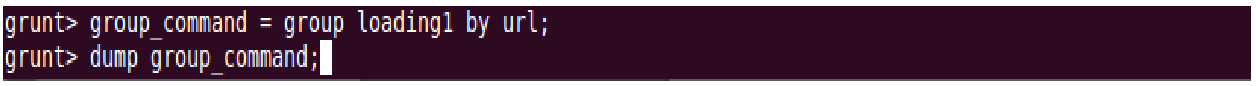
Goto : /storing

```
amr     cnn     8
amr     crap    8
amr     flickr  10
amr     myblog  10
fred    cnn     12
```

GROUP:

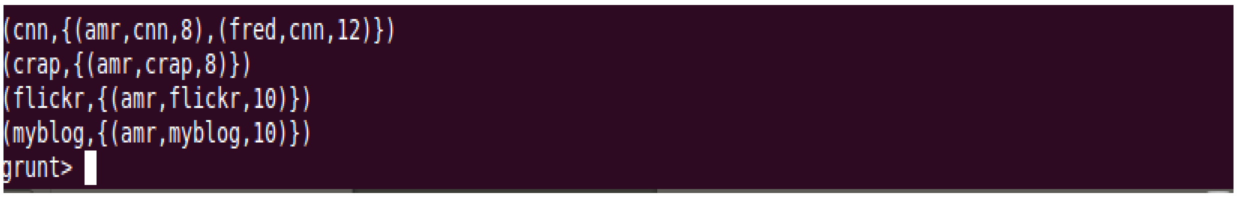
The GROUP operator groups together the tuples with the same group key (key field). The key field will be a tuple if the group key has more than one field, otherwise it will be the same type as that of the group key. The result of a GROUP operation is a relation that includes one tuple per group.

In this example, group the relation ‘loading1’ by column url.



```
grunt> group_command = group loading1 by url;
grunt> dump group_command;
```

GROUP Result:

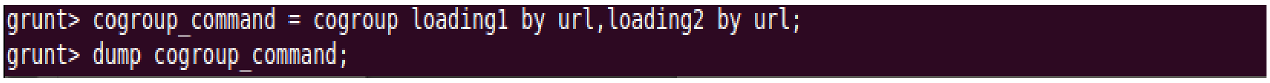


```
(cnn,{(amr,cnn,8),(fred,cnn,12)})
(crap,{(amr,crap,8)})
(flickr,{(amr,flickr,10)})
(myblog,{(amr,myblog,10)})
grunt>
```

COGROUP:

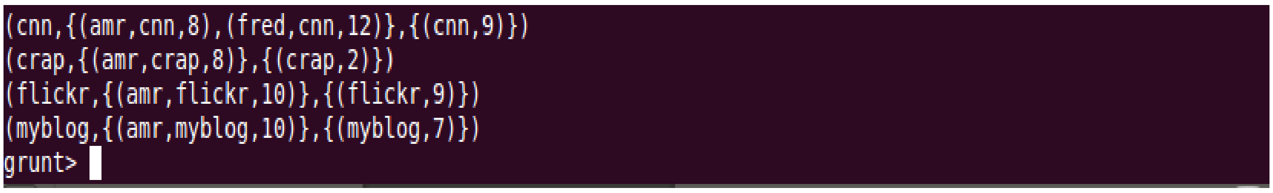
COGROUP is same as GROUP operator. For readability, programmers usually use GROUP when only one relation is involved and COGROUP when multiple relations re involved.

In this example group the ‘loading1’ and ‘loading2’ by url field in both relations.



```
grunt> cogroup_command = cogroup loading1 by url,loading2 by url;
grunt> dump cogroup_command;
```

COGROUP Result:

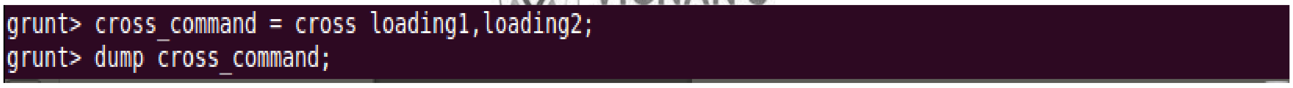


```
(cnn,{(amr,cnn,8),(fred,cnn,12)},{(cnn,9)})
(crap,{(amr,crap,8)},{(crap,2)})
(flickr,{(amr,flickr,10)},{(flickr,9)})
(myblog,{(amr,myblog,10)},{(myblog,7)})
grunt>
```

CROSS:

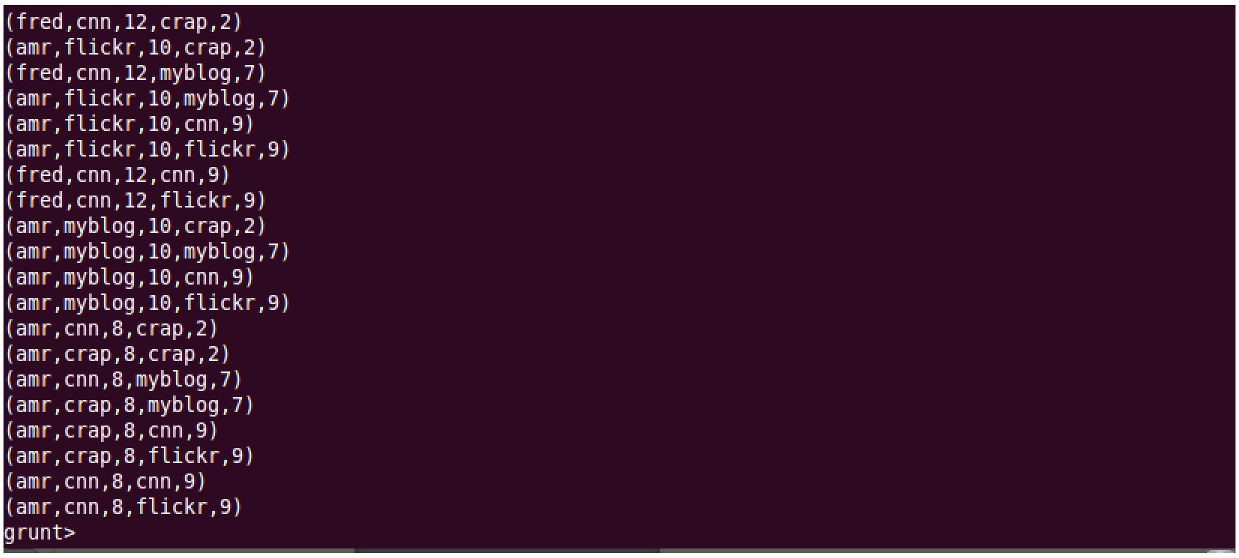
The CROSS operator is used to compute the cross product (Cartesian product) of two or more relations.

Applying cross product on loading1 and loading2.



```
grunt> cross_command = cross loading1,loading2;
grunt> dump cross_command;
```

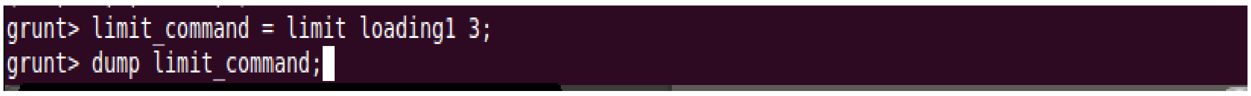
CROSS Result:



```
(fred,cnn,12,crap,2)
(amr,flickr,10,crap,2)
(fred,cnn,12,myblog,7)
(amr,flickr,10,myblog,7)
(amr,flickr,10,cnn,9)
(amr,flickr,10,flickr,9)
(fred,cnn,12,cnn,9)
(fred,cnn,12,flickr,9)
(amr,myblog,10,crap,2)
(amr,myblog,10,myblog,7)
(amr,myblog,10,cnn,9)
(amr,myblog,10,flickr,9)
(amr,cnn,8,crap,2)
(amr,crap,8,crap,2)
(amr,cnn,8,myblog,7)
(amr,crap,8,myblog,7)
(amr,crap,8,cnn,9)
(amr,crap,8,flickr,9)
(amr,cnn,8,cnn,9)
(amr,cnn,8,flickr,9)
grunt>
```

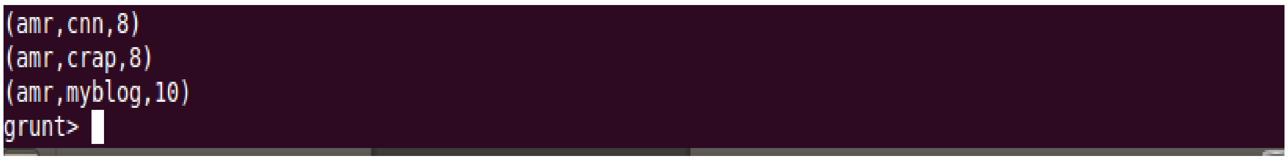
LIMIT:

LIMIT operator is used to limit the number of output tuples. If the specified number of output tuples is equal to or exceeds the number of tuples in the relation, the output will include all tuples in the relation.



```
grunt> limit_command = limit loading1 3;
grunt> dump limit_command;
```

LIMIT Result:

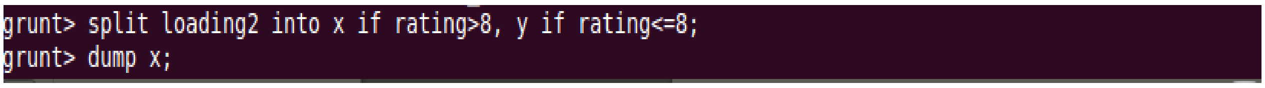


```
(amr,cnn,8)
(amr,crap,8)
(amr,myblog,10)
grunt>
```

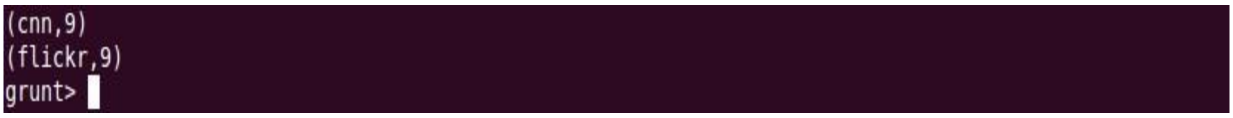
SPLIT:

SPLIT operator is used to partition the contents of a relation into two or more relations based on some expression. Depending on the conditions stated in the expression.

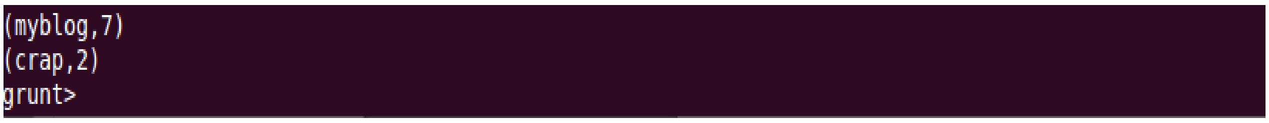
Split the loading2 into two relations x and y. x relation created by loading2 contain the fields that the rating is greater than 8 and y relation contain fields that rating is less than or equal to 8.



```
grunt> split loading2 into x if rating>8, y if rating<=8;
grunt> dump x;
```

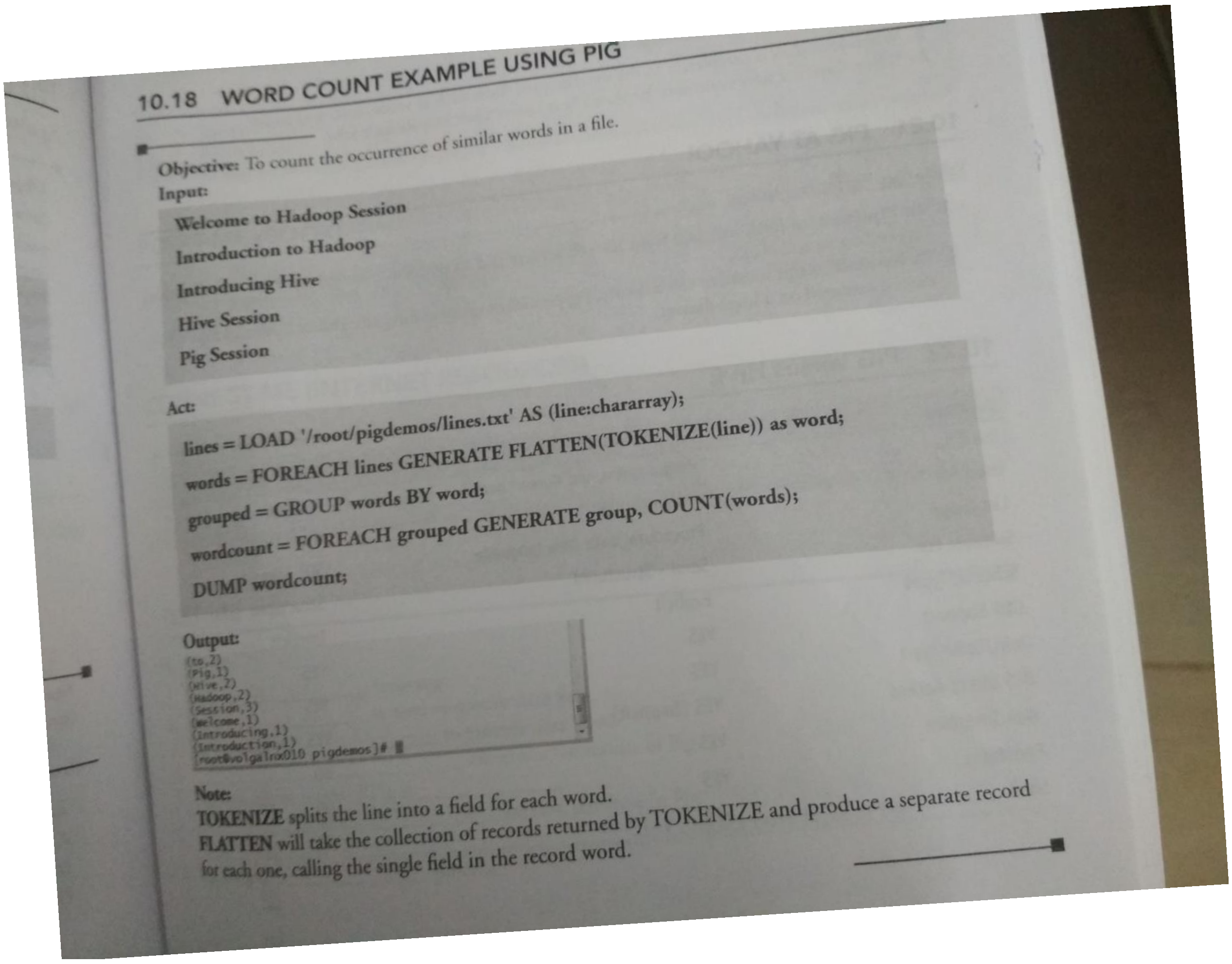


```
(cnn,9)
(flickr,9)
grunt> 
```



```
(myblog,7)
(crap,2)
grunt>
```

10. Implementation of Word Count using Pig?



10.18 WORD COUNT EXAMPLE USING PIG

Objective: To count the occurrence of similar words in a file.

Input:

```
Welcome to Hadoop Session
Introduction to Hadoop
Introducing Hive
Hive Session
Pig Session
```

Act:

```
lines = LOAD '/root/pigdemos/lines.txt' AS (line:chararray);
words = FOREACH lines GENERATE FLATTEN(TOKENIZE(line)) as word;
grouped = GROUP words BY word;
wordcount = FOREACH grouped GENERATE group, COUNT(words);
DUMP wordcount;
```

Output:

```
(to,2)
(Pig,1)
(Hive,2)
(Hadoop,2)
(Session,3)
(Welcome,1)
(Introducing,1)
(Introduction,1)
[root@volgalnx010 pigdemos]# 
```

Note:
TOKENIZE splits the line into a field for each word.
FLATTEN will take the collection of records returned by TOKENIZE and produce a separate record for each one, calling the single field in the record word.

11. Implement of word count using spark RDDs?

```
from pyspark import SparkContext
from pyspark import SparkContext
words = sc.parallelize (
 ["python",
  "java",
  "hadoop",
  "C"
 ]
)
words_map = words.map(lambda x: (x, 1))
mapping = words_map.collect()
print("Key value pair -> %s" % (mapping))
```

```
from pyspark import SparkContext
x = sc.parallelize([("pyspark", 1), ("hadoop", 3)])
y = sc.parallelize([("pyspark", 2), ("hadoop", 4)])
joined = x.join(y)
mapped = joined.collect()
print("Join RDD -> %s" % (mapped))
```

BDA VIVA QUESTIONS

1. **Define BDA?**
2. **What are the four characteristics of Big Data?**
3. **What are the Characteristics of DATA?**
4. **What are the types of digital data?**
5. **Examples of Structured data?**
6. **Examples of Unstructured data?**
7. **Examples of Semi Structured data?**
8. **Definition big data by Gartner?**
9. **What are the challenges with big data?**
10. **Whatare the application areas of Big Data Technology?**
11. **What does 'jps' command do?**
12. **What is Hadoop**
13. **How to restart Namenode?**
14. **What is a Namenode?**
15. **What is a Data node?**
16. **What is a job tracker?**
17. **What is a task tracker?**
18. **What is a heartbeat in HDFS?**
19. **What is a 'block' in HDFS?**
20. **What is Fault Tolerance in Hadoop HDFS?**
21. **Why is block size set to 128 MB in HDFS?**
22. **What happens if the block on Hadoop HDFS is corrupted?**
23. **How data or file is read in Hadoop HDFS?**
24. **How data or file is written into Hadoop HDFS?**

25. Which are the modes in which Hadoop can run?
26. What are the features of Standalone (local) mode?
27. What are the features of Pseudo mode?
28. What are the features of Fully Distributed mode?
29. What is the full form of fsck?
30. Which are the main hdfs-site.xml properties?
31. What are the benefits of block transfer?
32. What is the communication channel between client and Namenode/Data node?
33. What is a rack?
34. What is a Secondary Namenode? Is it a substitute to the Namenode?
35. Explain how do ‘map’ and ‘reduce’ works?
36. List some features of Hadoop?
37. What is Full form of HDFS?
38. Define HDFS?
39. Define MapReduce?
40. What are the HDFS Daemons?
41. What is Shuffling and Sorting in MapReduce?
42. What is Partitioner and its usage?
43. What are the main components of MapReduce Job?
44. What is the definition of Hive?
45. What are the components used in Hive query processor?
46. What is Buckets in Hive?
47. What are the different ways of executing Pig script?
48. What are the data types of Pig Latin?
49. What is Spark?
50. List features of Spark?
51. Full form of RDD
52. Full form of DAG
53. What is the role of Master node in spark?
54. What is the role of Worker node in spark?
55. Define Spark Core?
56. Define Spark SQL?
57. Define Spark Streaming?
58. Define MLlib?
59. Define GraphX?
60. Difference between map() and flatmap()?
61. What are the data types of Pig Latin?
62. What is a bag in Pig Latin?
63. What is UDF?
64. What are the different ways of executing Pig script?
65. What are the use cases of Apache Pig?